



The push registry enables MIDlets to set themselves up to be launched automatically, by setting an alarm or by sending data over the network. The push registry manages network- and timer-initiated MIDlet activation; that is, it enables an inbound network connection or a timer-based alarm to wake a MIDlet up. For example, you can write a workgroup application that employs network activation to wake up and process newly received email, or new appointments that have been scheduled. Or you can use timer-based activation to schedule your MIDlet to synchronize with a server every so often then go to sleep.

This is a general property of MIDP2.0 specification. But different vendors have implemented it differently. For eg; in Nokia & Sonyericsson phones there a permission message is displayed whether you want to activate the push registered application. Where as in Siemens phones only a star sign will be displayed.

Contents

- 1 The PushRegistry class exposes the following methods:
- 2 Exceptions
- 3 Using SMS Connections
- 4 Registration
- 5 Dynamic Registration
- 6 Static Registration
- 7 Unregistering the connections
- 8 To determine if the MIDlet was invoked by an incoming message:

The PushRegistry class exposes the following methods:

- `getFilter()`, to return the `<AllowedSender>` value for the connection (this can be a server IP address, a comma-delimited list of IP addresses, or an `*` to allow any connection).

- `getMidlet()`, to return the MIDlet name registered for the specified connection.
- `listConnections()`, to return a list of push connections registered for the specified

MIDlet suite.

- `registerAlarm()`, to register a timer-based trigger to launch the MIDlet, or to disable an existing alarm for the MIDlet if the argument supplied is zero.

- `registerConnection()`, which registers a connection for the MIDlet.
- `unregisterConnection()`, which likewise unregisters a connection.

Exceptions

The following exceptions should be caught:

- `ClassNotFoundException`
- `ConnectionNotFoundException`
- `IllegalArgumentException`
- `IOException`
- `SecurityException`

Using SMS Connections

It is useful to note that a MIDlet can initiate a socket or HTTP connection after it has been awakened by an incoming message, if further exchange of data is required.

The port specified can be from the full range 1 to 65535, however the following ports are reserved and must not be used:

- 2805 WAP WTA secure connectionless session service
- 2923 WAP WTA secure session service
- 2948 WAP Push connectionless session service (client side)
- 2949 WAP Push secure connectionless session service (client side)
- 5502 Service Card Reader
- 5503 Internet access configuration reader
- 5508 Dynamic Menu Control Protocol
- 5511 Message Access Protocol
- 5512 Simple e-mail Notification
- 9200 WAP connectionless session service
- 9201 WAP session service
- 9202 WAP secure connectionless session service
- 9203 WAP secure session service
- 9207 WAP vCal Secure
- 49996 SyncML OTA configuration
- 49999 WAP OTA configuration

Registration

Push Registry can handle requests to register connections in two ways:

1. dynamically at run time,
2. statically through entries in the JAD file

Dynamic Registration

Dynamic registration is a MIDlet notifying the AMS at run time that it wants to be activated by an incoming network connection or alarm event should the MIDlet be exited prior to that event occurring.

For a connection the registerConnection method is used:

```
registerConnection(String connection, String midlet, String
filter)
```

and

```
this.getClass().getName()
```

can be used to specify the current MIDlet.

Some of the examples for Dynamic Registration :

```
registerConnection(?sms://:? + portNumber);
registerConnection(?datagram://:? + portNumber);
```

Static Registration

If a connection's sender and connection type are known when the MIDlet is installed, the registration request can be made at installation, and is therefore regarded as static. Static requests are defined in the JAD file using the Midlet-Push-<n> attribute:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>,
<AllowedSender>
```

where n is a sequence number allowing more than one connection to be declared, ConnectionURL is the URL to monitor for an incoming connection, MIDletClassName is the MIDlet to start, and AllowedSender is the filter: a list of IP addresses or * for any.

An example for SMS connection :

```
MIDlet-Push-1: sms://:10000, TestMIDlet, *
```

Unregistering the connections

Dynamic registrations can be removed by using unregisterConnection specifying the connection only:

```
unregisterConnection(?sms://:10000?)
```

To determine if the MIDlet was invoked by an incoming message:

In startApp():

```
String connectsFound[];
```

PushRegistry

```
connectsFound = PushRegistry.listConnections(true);
```

This returns the list of registered connections.

```
if connectsFound == null || connectsFound.length == 0)
{
~ started by user, code to exit or bypass push-related activity ~
}
else
~ started by inbound connection so code for push registry initiation ~
}
```

Sun Resourece: <http://developers.sun.com/techttopics/mobility/midp/articles/pushreg/>

java world resoure: <http://www.javaworld.com/javaworld/jw-04-2006/jw-0417-push.html>

Sonyericsson resource:

http://developer.sonyericsson.com/site/global/techsupport/tipstrickscodes/java/p_tips_java_1201.jsp